TASK IV

Action Languages:

Dimensions, Effects

A REPORT OF THE YEAR'S RESEARCH
TO THE NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
MARSHALL SPACE FLIGHT CENTER

NCC8-13

Daniel G. Hays, PhD
Task Leader
Gordon Streeter
Associate

Johnson Research Center
The University of Alabama in Huntsville

Research Report No. 805
August 31, 1989

Table of Contents

I. Introduction 1
1. Research Goals and Scope 1
2. Research Funding 3
3. Personnel 3
II. Overview 5
III. Action Languages 6
1. Goals 6
2. Language and Action 7
3. Dimensions of Action Languages 9
4. Behavior of Ambulatory Devices 12
5. Language Characteristics: Some Issues 19
6. Effects 21
7. Communicating About Effects 23
8. Talking, Doing, and Showing 24
9. Talking to Machines 26
IV. Dynamic Communication Systems: Message-Passing in Object-Oriented Languages 32
1. Introduction 32
2. Scope 33
3. Object-Oriented Languages 34
4. Using Existing Implementations 37
5. Extending the Dispatcher 41
6. Conclusion 44
V. Conclusions and Issues 45
References 49

Action Languages

a report of the year's research
to the National Aeronautics and Space Administration
Marshall Space Flight Center

Dan Hays, PhD, Task Leader
Gordon Streeter, Associate
The University of Alabama in Huntsville

I. Introduction

1. Research Goals and Scope

This document reports an investigation of what we have chosen to call *action languages,* the means of communicating about behavior in situations. During this project we were especially concerned with the "behavior" or events associated with mechanical or electronic devices. The basic concepts are more general. Action languages could refer to the behaviors of animate creatures as well as machines. Indeed, even for device-action languages, though we often imagine a person telling the machine what to do and the machine silently and eagerly complying, a more workable arrangement would also involve communication from the device to the person, in the nature of feedback, requests for human intervention, and so on.

A related topic examined during the year was *capabilities of present-day object-oriented computer languages.* In a sense, the message-passing schemes of object-oriented programming systems (conventionally

abbreviated as OOPS) constitute one kind of action language, though operating just within computational devices. Further, the OOPS provides one model for action languages in general in that real actors can be viewed as "objects" that pass messages with the intention of generating actions. Besides this, we wanted to examine existing object-oriented systems for adequacy. In particular we were interested in the question of discretionary response to the "messages". Present-day "objects", it turns out, even when they are called "actors" or "agents"[1] are not usually set up for discretion, though it may be possible to program them to evaluate what they are requested to do before they do it.

Theoretically, action languages could be designed for both simple uses and more extensive or complex ones. As some of the discussion will suggest, even simple cases can become fairly complex when the realities of action situations are taken into account. In other cases, such as that of the so-called Command and Control Languages, communication conventions sometimes seem to have been simplified to fit a lean view of situations. By focussing on communication about action, and the situations in which actions occur, we would like to call attention to practical requirements for adequate information exchange.

The subject matter of the research has thus been fairly basic. It was also deliberately programmatic. This year's work was cast as the first of three years of ever more specific exploration of the action language concepts. So, the ideas presented in this document are in a sense preliminary to more specific development that as it turns out might be pursued at some other time, in a different context.

During this year, certain basic issues.of action-related languages were raised. In addition, attention was given to the kinds of situations that

[1] See Tello (1989), Chapter 9; Hewitt (1977), Hewitt and DeJong (1983).

Space-resident devices might perform in. This material suggests some of the complexity of the design issues that may well be involved for human-machine and machine-machine communication, if one accepts the principle strongly urged here that languages for machine communication should usually include facilities for describing environmental features and dynamics, as well as 'blind' machine movements.

2. Research Funding

The Action Language task was one of several included in a Cooperative Agreement between the National Aeronautic and Space Administration's Marshall Space Flight Center and The University of Alabama in Huntsville. The title of the larger project is "Foundations of Automated Software Techniques". This research is listed as Task IV of the agreement.

3. Personnel

Dan Hays, PhD, served as Task Leader. He is Associate Professor of Psychology and a researcher at the Johnson Research Center of the University of Alabama in Huntsville. Mr. Gordon Streeter served as research associate, technically a Consultant, on the project.

Though discussion of various issues has been shared, Hays has been primarily concerned with the conceptual analysis of situated action languages, and Streeter with evaluation of object-oriented computer systems.

The role of various people at UAH who are concerned with research on artificial intelligence and knowledge systems is also acknowledged. Particularly helpful was discussion with students from various states who participated in a summer program funded by the National Science Foundation

of which Hays was Project Director. This program was titled "Knowledge Organization for Machine Systems". The summer of 1988 was the second time that it was held.

The role of the Johnson Research Center of UAH must also be acknowledged, both for its specific role in this research and also for having supported the development of machine intelligence work as an interdisciplinary pursuit at the university.

During the early months of the project the energetic participation of Mr. John Wolfsberger, then still a Marshall Space Flight Center employee, is gratefully noted. He was very helpful in making explicit the kinds of existing problems and potential applications that inquiry into device action and language systems might impact. These discussions often centered on the production and coordination of software–especially software for the testing of complex devices.

For the Cooperative Agreement which funded the research reported here, Donnie Ford was Principal Investigator at UAH during the first months. Jim McKee assumed these duties during the latter part of the project year

II. Overview

This report covers these main topics:

- *characteristics of action languages,* including major dimensions and relation to situations,
- an *evaluation of current object-oriented computer languages,* especially for their capability to handle discretionary actions,
- summary comments.

During the course of the report, various *sample action languages,* or parts of them, will be considered, to help tie the discussion to concrete problems.

Because of the fairly basic nature of this research, we were concerned more with working out conceptual issues–and simply coming up with the ideas that would be involved in working with action languages and extensions to object-oriented computer programming systems–than with the development of applications. However, it should be noted that sample computer code is included in the discussion of object-oriented languages. Besides illustrating points of the report, it is meant to suggest what code might look like for later development.

III. Action Languages

1. Goals

The *involvement of language in action* is the major concern of this of this project. Just how communicated signals can be interpreted and related to activities that accomplish something, is the basic question that has been explored. The focus has been intentionally broad, so as not to obscure issues that may be important in the design of realistic action systems.

Sometimes the language-action relation is fairly straightforward, at least in principle, for example where predetermined signals are supposed to trigger planned events in relatively static systems. Other cases may introduce considerable complexity in that they involve *discretion in the interpretation of information.* Again, certain tactics of communication need to take into account that *situations can change quickly,* so that the communication must be sensitive to local conditions. In these cases, information and decision-making are often decentralized. In such situations, to be sure, some of the parts or participants may behave in a simple way or have only limited ability to "think" about what they are doing. The situation itself though may be complex in terms of actual events, or—more critically—in terms of the possibilities that must be considered by the system designer.

Providing conceptual analysis of the situations of language use is the task at hand. The analysis reported here centers on

- dimensions of 'action languages', with special attention to
- the way that *effects* occur in action systems.

Always in the background is the question of *desiderata* for programming systems and other arrangements that would allow useful and informative communication in action situations.

2. Language and Action

An action language is one that *directs or modifies the performance of actions in environments*. Specifically included would be not just directives, but also background information and descriptive messages that could at some time be relevant to actions.

The definition is intentionally broad. It allows inclusion of languages that tightly direct performance, such as ordinary programming languages for computers, conventions for "command and control" situations, robot languages at various levels of specificity, and so on. Languages with less direct relations between what is said and what is done are also included, such as specification-based computer languages or ordinary human languages such as English.

English—or Japanese, Russian, Hausa, etc.—do a lot more than just referring to action and directing it. Present computer languages do a good deal less, except when dealing with the exact devices and very restricted situations that they have been developed for.

Goals. Though the span of linguistic and situational interest has been kept broad, since a basic understanding of language-action relations is being sought, the real concern of this part of the project is with action languages that are specific enough to apply to *capable machines in their interactions with one another and with human beings*.

Underlying the general concern for the language-action relation expressed here are the more concrete but difficult goals of designing language conventions that are adequate to the next generation of smarter, more capable and mobile machines.[2] The language conventions for capable devices probably also have to allow for *informative communication with their human users,* though cases can be imagined where machines are interacting in situations so remote that ready communication with humans would be beside the point.

The examination of capabilities of object-oriented programming systems (OOPS) for current computing machines, reported in a subsequent section, is germane to the issue of easy communication means and manageable programming. Of software technologies currently available, object-oriented techniques seem to offer the kind of modularity and referential capabilities that might serve as the basis for more complicated systems.[3]

Another area which can be treated linguistically, in a certain reduced way, is that of *intra-device communication,* where parts are given a rudimentary ability to detect certain conditions (e.g., pressure, abutment) and to communicate appropriately about them.

The conceptual inquiry is seen as necessary in *developing some of the concepts that will be helpful in planning communication conventions for devices and people* that interact in situations of physical action.

The language-action relation is a broad and complex one. Some important topics involved in it are:

a. *dimensions of action-oriented languages,*

---

[2] Good action language conventions and capabilities for existing devices would be useful also.

[3] Though today's object-oriented languages could stand some development, as we discuss elsewhere.

b. the *kinds of actions* that machines might be involved in,
c. *situational aspects* of action and action communication,
d. the *effects of actions,*
e. *human needs and biases* in communicating with machines.

3. Dimensions of Action Languages

A language is a set of signs that have some meaning or reference. "Set" may be too sparse a term, since the signs used in a language generally have an identity as belonging to the language, and some kind of restrictions on contexts of usage.

In this discussion, the viewpoint of general *semiotic theory* is taken. This field discusses sign systems of many sorts, not just ones of ordinary human verbal language but also the languages of signs, clues, symptoms, and so on in various media.

The coherence or belongingness that we sense in the parts of a certain language (whether it is the set of conventions that we use to prove theorems in an algebra, or the "language of flowers") seems to have to do with the organizing properties of the *system of interpretation* of the language. This is the set of rules, or perhaps procedures, that allow those familiar with the language to discern the patterning of signals (syntax) and to relate the signals to meaning (semantics or pragmatics).

Ordinarily we think of the 'signal' part of signs[4] as being separate from what is being referred to, and as arbitrary. In ordinary discussion, for

[4] In semiotic theory, the term "sign" usually refers to the complex of signal and referent or interpretation, and is distinguished from signal or physical carrier of the language's messages. However, because of ordinary English usage of the term, "sign" is sometimes used to focus on the language token or figure apart from its reference. The theoretical distinction should not be troublesome at the level of discussion of this report.

example, we speak of *symbols* as being items that have in some abstract way come to refer to something special. This is the case for much of spoken or written human language, where sounds of a certain pattern bear no intrinsic relation to what is being talked about. But not all signs are so abstract. Certain signs, called *icons* by semioticians, are similar in some way to their referents. Graphic gestures and road signs are often iconic. Another class of signs, called *indexes* or *indices* in the jargon of semiotic theory, are more closely involved with what they are referring to. A *symptom* (whether of a common cold, or a healthy economy) is a kind of index. The signs of nature, which are themselves part of nature, are 'indexical' in semiotic terminology. Note that an indexical sign depends on an interpretive system for both determination of meaning and communication. For example, the same crack in a structure of rocks may be seen by a geologist and a lay person, but probably only the geologist is onto the system of interpretation that gives the meaning and implications of the fault indicator. Indexical processing is especially relevant to action languages, as developed here, because of the important role of perceived ongoing action.

Language and Action. Languages that deal with action may be capable of one or more of the following kinds of functions, though some languages are probably more tailored for just some of the following. A fully capable action language should handle all of the following. One can distinguish among:

- *descriptions of action,* perhaps also including related causal explanations for what is done;

- *directives for action,* that with some degree of forcefulness suggest, channel, command, or even coerce behavior;

- *ongoing concomitants or modifiers of action,* communications that are closely integrated into the behavior.

- *discussions of action* among potential participants, advisers and directors, either in advance for planning, or retrospectively, to evaluate what has been done.

Biases of language definition and focus. Though all of these seem quite naturally related to action, and desirable to have in a language, it seems fairly common for people to focus primarily on just one.

We mention this here since it is common for people to think of "language" in restricted ways. We are arguing for some breadth in the application of the term to action situations.

One bias often found among scholars, and possibly even in the whole tradition of Western academic thinking, is to be concerned primarily with *descriptions* and associated explanations, evaluations, and so on. (There may even be a tendency among academics, from Plato on, to shy away from action descriptions in favor of more abstract terminology of qualities, virtues, tendencies, essences, and so on.)

By contrast, most computational work relating to action systems (e.g., manufacturing, military, or transportation devices) concentrate heavily on *directives* for action, with descriptive material playing at best a secondary role. The same bias may be seen in managers and engineers. Plans for action, especially of machine systems, are often so concretely determined, in detail, that they are virtually identical to strings of commands, with specifically planned contingencies.

Action agents may in some cases care little for description and explanation, or yet again for close direction, but will be involved in the initiation and ongoing modification of behavior in real situations. Action becomes real for the actor, and the perhaps unexpected contingencies of the environment are not only real but may be surprising. For many agents, of course, the description or explanation of intended behavior may be critically important. It is probably a good principle that the more intelligent and autonomous the actor, the more that descriptions will be relied on, especially in communication with others who are concerned with the behavior.

Various groups may be concerned with plotting future action or with retrospective analysis of behavior that they or others did. Often, this is regarded as background or workshop activity, outside of the main arena where action occurs.

It is the heavy focus on minutely planned, specific directives for action, so widely found as an assumption and practice in the technical and managerial fields when planning for action systems involving devices[5], that we believe needs close examination and challenge. Action itself is always specific, whether specifically planned or not. But, detailed external specification of behavior may not always be a feasible approach. This is true both of descriptions and directives, which may be similar in some cases.

4. Behavior of Ambulatory Devices

Consider what a capable machine might be expected to do in a remote environment, or one not so remote, and what it might need to process in

[5] Or, for that matter, involving the actions of people. One might compare the detailed protocols prepared for the actions of Mission Specialists on Shuttle missions, for a striking example of detailed directives for behavior.

order to function adequately.

Various devices of interest to the Space program might be expected to:

1. Successfully maneuver over ground or in some fluid medium.

Such locomotion in any medium

- requires the processing of various subtasks,
- requires orientation to environmental patterns,
- requires matching of environment to directives or other goals
- may require attaching to and coordinating locomotion with other devices or a *matrix device.*

2. *Relative to objects or devices treated as unintelligent,* the capable device might have to

- plug into and dump electronic readings, or send probe signals,
- stabilize,
- adjust knobs or other settings,
- remove fasteners,
- check connections,
- jostle,
- sense state nonelectronically (by processing patterns of sound, vibration, color, position, etc.),
- secure to an environmental holder,
- secure to transportation device (including self),
- transport (see comments on locomotion above),
- other conceivable kinds of action.

3. Coordination of actions with other similar action-devices might also be required. This could involve:

- locomotion:
  - co-orientation,
  - assistance in travel,
  - one moving, one watching, etc.;
- working on an object or device:
  - one stabilizing, one operating,
  - joint action on object or device;
- assistance in self maintenance or repair.

4. Relative to environmental features or regions the device might have to
- sense and scan,
- rearrange the environment, by
  - digging,
  - boring,
  - moving separable parts of environment into configurations,
  - adding and arranging parts brought in.

(See also locomotion (rearranging self relative to the rest of the environment.)

5. Relative to other intelligent devices (see also, humans) the device might need to
- identify them,
- report information to some of them
  - re environmental standing features,
  - re environmental transitory features,
  - re environmental possible features,
  - re state of self,
  - re organizational superstructure,
  - re tasks directed elsewhere or judged to be desirable;
- request or direct other action,
- request or direct coordinated action,

- assess cognitive state of the other devices in making judgements about the feasibility of interaction

6. *Ordinary computation might also be required.* The device might
   - calculate or compute abstract information,
   - serve as ordinary storage-retrieval device for factual databases,
   - and so on.

Processing Required for Action. For the kinds of ambulatory devices that one might wish to have, the processing requirements are certainly formidable, though considerable work is being done in many laboratories to work out procedures and understandings necessary to build such devices.[6]

At a generic level, the following kinds of things would be required:

The machine has to have adequate input reflecting the state of the environment in order to get around.

The actions of the machine have to be adaptive to local conditions, including unanticipated or unremembered small details.

Generally useful programs for doing things like locomotion in the usual medium should be readily available to the device or device system. These should function concurrently but in some kind of communication with some of the other processing, but work reliably at a very deep level of the system.

At the same time that these generally useful programs are in operation, together with associated memory/knowledge operations, the device has to be able to access

[6] Or, to build them better, since ambulatory devices, and various capable stationary devices already exist that do some of the things mentioned, at some level of competence.

- a variety of cue-patterns that, if detected or approximately detected, could either be
    - stored away as information about the scene, or
    - used for immediate replanning of ongoing actions.
- computation that assesses when a certain level of problems in carrying out actions means that reevaluation of more molar actions must be done
- ordinary, ongoing attention to energy levels, resources, etc.
- possibly, attention to coordination of action with other units.

If a machine is walking (or swimming, flying, or being propelled) someplace, it needs to have a stock of small component plans for getting around obstacles, adjusting to buffeting or signs of danger, and so on. Perhaps more importantly, it needs major adjustable locomotion and maneuvering capability. Some of the plans for exceptions or auxiliary activities could feed into the locomotion/guidance 'engine' from separately computed sources within the device. The question of how to balance motive control/adjustment computation distribution and input from exteroceptors or external sensory systems, planning referents, and other processing of distal information is certainly an interesting one.

If the machine is braced someplace, fixing something, it needs a somewhat different mix of activity. Needed will be:

- computation to maintain its braced position (or to counterbalance moves with propulsion or some such),
- adjustable movement computation, communicating heavily with the brace/stability computation, for its efforts in manipulating things,
- superordinate task planning and execution routines which themselves have backup heuristics (e.g. if a bolt sticks) and ongoing sensitivity to things that are noticed while performing the maintenance,
- possible communication with other devices
    - working with it, or

- in a supervisory or informational position;
- ongoing but occasional abstraction and storage of information about what it encounters in working on the task, or notices incidentally.

The above implies a very large amount of computation, as well as sensory capability and control mechanisms for movements. Perhaps some of the functions could be integrated into the major computation and control for basic activity, though multiple computation centers, perhaps heavily distributed ones within the device, may be required.

Effectors and perceptual arrangements require essentially continuous attention at some level. Other computation could be more occasional, for example to evaluate overall task progress.

Additionally taking into account signals from other sources, and generation and giving these signals for coordinated action, adds further challenges to the design. Yet, coordinated action would certainly be desirable in many cases.

Thus, the rather complex computation for action in a situation is a *major context into which an action-related language must be embedded.* The sending and receiving of signs, their interpretation, and possible subsequent behavior adjustment, must fit smoothly into the computational complex of the action devices which are participating in the interaction.[7]

More Modest Devices. To build capable ambulatory robots and to design procedures for making them work smoothly and effectively are real goals for many people. More restricted devices may also participate in action related signal evaluation.

---

[7] Or possibly, those that are observing the action scene and evaluating it.

Mentioned above was the possibility of designing small "languages" for parts and subassemblies of larger structures.

These would involve such structural features as

1. Partitioning of certain physical parameters, perhaps by mechanical means, into a small set of values, each with an associated symbol or value within a syntactic category;

2. A typology of kinds of states relevant at the "part" level. Dichotomous or small-valued discrete symbols could probably handle much of what would be needed: a ternary logic would seem to provide a lot of structure. Such features as abutment/nonabutment, attachment/ disattachment, valve closed/possibly open/open, acceptable level/marginal level/unacceptable level of *v*, stored/discharged/uncertain, whole/severed, and so on.

3. Patterns of features could form a small corpus of potential sentences processed locally, which would then map into transmission states.[8]

4. Connectedness operators, or some such, could mediate local, and eventually more extensive communication. Such communication could be initiated externally (as in polling or external imposition of interpretation) or from certain conditions at local levels.

Other kinds of devices, and device communication arrangements might be analyzed. For example, it is a very common pattern to set up a kind of *reduced language relating a small set of discrete signals to device events.* Such languages would probably have a minimal syntax, e.g., unitary commands strung out with constraints dictated by actions taken rather than

[8] One images both local processing, and even local means of processing (what was referred to in an earlier report as a *local logic*).

linguistic structures or contextual sensitivities. The sign-act linkage may be done routinely, with selection dependent on pre-wired state-sensing or other determination, as in various automated manufacturing situations; or with interpretation 'modules' involving human consideration, as in some of the spectacular long-distance device control missions of the American Space effort.

Often in these token -> preset action sequences there is a one-sidedness of control, as well as a kind of event conceptualization that forces predicting a presumably exhaustive tableau of scenarios within the scope of activity of the device and its often simplified environment. Such careful engineering is often essential, since little real intelligence is built into the machines (though their form may reflect real ingenuity in design), and costs of failure are very high.

In more fluid situations, however, or ones where remoteness or danger precludes human mediation, the action structure needs to be more differentiated, and more evaluation needs to be made in the situation. Chances are good that such evaluations will not be directly related to actions, but will need some kind of structure, perhaps at about the level of an "expert advice system".[9] to approach the problems well.

5. Language Characteristics: Some Issues

One of the problems in conceptualizing action-oriented signaling systems involving devices is that there are many kinds of devices.

Generally, we build machines to solve certain kinds of problems, though we may later discover that the machine or tool also can be applied to other

[9] Increasingly familiar as a fairly straightforward way of representing the evaluation of input for decision-making and initiating search for more adequate information when needed.

problems that we did not think of in the first place.

. Machines that we make tend to be specialized–even computing machines, which have a reputation for generality–and they vary greatly in complexity.

So, *the extent and nature of a language for machine action will depend on the greatly variable nature of machines.*

Indeed, we should design the machines with communication systems in mind.

This means that some devices will have very restricted signaling, perhaps only serving as indexical signs for sentient systems such as humans or observing robots. Other devices may have a very "mechanical" appearing set of state tokens to communicate about. Still others must be designed with substantial ability to perceive and evaluate dynamically changing situations.

It is difficult to say that a sensor has real understanding of a situation, but it would be difficult not to aim for good "understanding" by a capable ambulatory device that is intended to respond to relatively novel events.

For such capable devices, it seems clear that

Understanding is more basic than linguistic communication.

By "understanding" is meant here a kind of "action knowledge", or ability to respond in a sensitive way to changing situations.

In such cases, the interpretation and evaluation of messages from others would be *just part of the interpretation and evaluation of information in*

the situation of action.

6. Effects

Traditionally analysts been heavily interested in causes. This seems generally true of a fair amount of philosophic inquiry and much other academic discussion. It is a clear characteristic of scientific inquiry, and is pervasive in applied fields of diagnosis of problems of built systems.

The study of action languages leads to a concern for effects. Perhaps this is the reason that they lend themselves readily to talk of the design and implementation of systems. "Action communication engineering", concerned like most other sorts of engineering[10] with effects

In this section, the structure of effects will be discussed somewhat briefly. If causes are sometimes tricky to tease out and verify, effects are often widespread and unwieldy to trace, especially if secondary effects or side effects are examined.

Relative both to causes and effects, the conceptual focus of the analyst seems very important. One investigator may, for example, only be looking for certain kinds of causes, labeling the others as extraneous to an investigation. Similarly, a person may be looking only for certain kinds of effects. Whether or not one should limit one's view of effects is debatable, especially for potentially dangerous systems.

---

[10] The term "engineering" is used in a broad semantic sense, and should not be taken as implying something taught in engineering sch ools. Besides concern for effects, engineering is also of course concerned with causes—it is difficult to be concerned for one without thinkg of the other. But the difference of *focus* on cause versus effect is not trivial, it seems.

Factors Important to Effects. Various factors would have to be examined in characterizing various effects. These include:

- Locus a given effect, both its *initial locus* (if appropriate) and *subsequent identifiable loci of changes.*

- Timing of the effect. This could simply be
  - when it occurs, in a point sense, or
  - sequenced effects,
  - temporally facilitated effects, including levered effects and buildups,
  - other temporal relations.

Nature of the change, whether to material, covering, connectedness, etc. Note that the effects of some events is to halt or impede change, however.

Simple effects, versus articulated or component effects (could depend on nature of what is being affected).

The cases of effects on a moving system, or on a system already in some process of change, seem especially difficult to conceptualize.

There may also be non-focal effects, that is on objects or environmental parts that are not of primary interest, but that might be useful to know about at some time. For example, if a rocket is fired toward a destination point, changes may be made in the atmosphere that it passes through that could matter in some way.

In this connection, the *solidity versus diffuseness* of effects is interesting.

Action Granularity. The "granularity" of action, a metaphor, has to do with the fineness of detail, either in a physical or a systematic sense, and relates to just how an action can be effective.

To some extent, granularity may just be a choice of the analysis, but there could be a kind of *effective granularity,* reflecting at what level of a system the organizing effects actually come from.

Finer distinctions of action type need to be added to flesh out the concept. For example, one could simply analyze fineness of detail of action and effect. But if one also says what kind of action is involved, more information is given.

7. Communicating about Effects

In an action situation, it is usually effects that we are concerned with, though we may sometimes be concerned with propriety (even apart from the effects of being proper or not).

In an action language, communications may be aimed at

- identifying worthwhile effects, so they may be sought,
- identifying negative effects, so they may be avoided or neutralized,
- examining effects to determine their nature,
- exploring, advising, or urging actions that would lead to effects.

There are variations on these, such as participating in action patterns designed to limit damaging effects.

A major distinction relating to communicating is *when it is done relative to a given action.* It will make a difference both in aptness and in form of communication whether the message is delivered well in advance of

the action, as a kind of plan, or is delivered in mid-action. The message may be late, or could be a post-action evaluation, also.

An interesting feature of action languages is that the processing demands of the situation of reception, and possibly of transmission, may affect the form of the action-related message. For example, more leisurely descriptions can be discussed in advance on an action, or thereafter, though length may be tedious (in less human terms, cause processor overload) in any case. In the heat of fast-moving action, a message will have to be terse at best.

8. Talking, Doing, and Showing

By now, enough options for behavior and possible dimensions of situations have been introduced to suggest that, if action languages themselves might not be too complex, at least the choices made in restricting them may be difficult.

Even so, communication conventions for the behavior of devices are currently pretty simple from a structural point of view, if we consider external actions of machines. (Computational devices per se have complex languages, though little is left to the discretion of the computers in many cases.)

Present-day robotic devices are almost always involved in communications loops, but the communications involved are formally fairly simple. This is true because even now most such devices are treated as blind machines, that is systems that can move is some limited space. Relation of the movements to things is often unspecified, though as the technology of force-sensing and other sensory coordination improves, environmental references in the robotic languages should also be expanded.

But this is a problem, because of what seems to be a *blindness bias* in

the field. The situations in which most industrial robots operate is of course quite constrained. So that if a part to be worked on is in place, the coordination of the part's locations and the robot's locations can practically be a responsibility of the matrix system, e.g., the assembly line arrangement, with only minimal sensory information from the robotic device itself (in many cases there is none at all).

An action-specifying language for a blind device is not necessarily a simple language. Technically, if movement within a restricted space can be ordered up and these movements have real-valued coordinates, an *unlimited number of actual patterns of behavior* can be generated by the system.[11] The effects, though, may depend on what is in the environment. Nothing might happen, or harmful and unplanned events could take place, depending on what else is there besides the machine, whether environmental entities are flexible or rigid, sentient, etc. So, an infinitely large set of behaviors (movements) could ensue, and none of them would involve any sensitivity to the environment.

It is difficult both to describe the environment and further to coordinate action with environmental knowledge (or supposition). But this is necessary. What may also be necessary before machine communication conventions advance very far is to design machines for somewhat higher level computation, and a bit more autonomy.

As an example of approaches to robotic languages, consider the reports in the volume edited by Ulrich Rembold and Klaus Hörmann, *Languages for Sensor-Based Control in Robotics* (1987). Only a small number of these papers exhibit anything that looks like a language, and these are fairly simple (start, stop, move(.), grasp). The papers are primarily concerned with procedures for integrating sensory information, or the details of the math involved in the control of the effectors. Though to speak of "language" has a certain currency, the matters addressed in that volume, and in many

[11] This is reminiscent of the capability of languages to generate indefinitely large numbers of sentences.

other places, do not have much to do with linguistic possibility or necessity, but with the mechanical or mathematical background that would be referred to or that would be assumed by a system of communication.

9. Talking to Machines

In considering the means of communication with a device several questions are important:

What needs to be communicated?

How would we *like* to communicate with our devices?

And, what do our devices need to tell us?

The answers to these questions are practical ones, and may be particular to situations. But the questions are not always asked. Frequently, the communicative aspects of machines are not specifically designed (though the sensitive designer will certainly consider this aspect). We think of machines as doing, not so much giving information. But devices in action always give off some kind of signals–not necessarily the most informative ones. Other devices are designed specially to give limited messages, in a message-leaving place. At other times we do not need direct communication with a system, in the sense of messages exchanged and interpreted by both parties. We can get all the information we need just by observing the machine in action.[12]

The same could be true of a sufficiently sensitive and intelligent device in its relations with people (or other devices). Just as we guide much of our own motor action by observing what is happening, rather than because of detailed verbal directive, a sufficiently adept machine should rely very much on ongoing situational understanding. In such a case, communications would only be part of a larger scene of action, and processed along with the other sources of information.

[12] In semiotic theory this is considered to be meaningful but unintentional communication. A major class of signs called indexes are involved. An index is a kind of natural sign, such as an event of nature, a symptom, or a clue. The system of interpretation is imposed by the observer, but the causation of the sign is not in general assumed to involve volition.

Related design decisions involve how much we want to be automated, and how much we want to be passive, how much of a system we want to be sensitive, and how much should be structurally useful but stupid.

There may certainly be tradeoffs between having parts that are sensitive and parts that are structurally sound in a mechanical sense. Further, if there is one glaring fact about many sensors it is that they may be unreliable.

Despite the bias of at least the senior author that robotic devices should be massively sensitive, and as cognizant of their spheres of action as their resources can allow, many successful robotic operations today are blind, tightly controlled, dependent on close environmental structuring, and so on. These are all design choices–though it seems that in the world of robotic design, the side of blindness and external control has been much more explored than have possibilities for semiautonomous and sensitive action devices.

How we would like to communicate. If we take seriously the second of the above questions (how would we like to communicate with machines), we will probably find that we want to communicate less rather than more, and that we would like for our machines to understand what we want to know about or for them to do without detailed explanation. Of course, we want them to carry out actions compliantly and competently.

Suppose we have a system with several valves which permit or block the flow of fluids. What kind of Valve Language would we like to have, to communicate important things to the system?

The simplest case involves no automation. We just observe whether or not a valve is open or shut, or partially so, using indexical signs: the position of a handle, the sound of the fluid in the neighborhood of the valve or its connected tubing, etc. If we can physically open and close the valves, we are giving a kind of pragmatically effective "signal", in a certain sense.[13]

In the next most complex case, there exists some way that we can get information about the state of the valve, or the fluid flow in its neighborhood. In monitored systems, such information is important, whether or not there are remote means of opening or closing the valves.

We can get signals purporting to give information on valve state (say, closed, open, partially open, or some finer gradation) from various kinds of proximal sensors, or from inference made from slightly remote sources. (If the fluid got here, that valve must be open.) If there is a sensor or sensor combination that can be treated as a unit at the valve site, it will respond with one of several signals indicating its state of openness. It may volunteer the information, or be asked. If we, or a supervisory unit, wants to know about the state of the valve, a question can be asked: Are you open or not?

We might also want to know such things as: Are you open, and is any fluid passing through? What is the rate of flow (temperature, etc.)?

At a perhaps more subtle level, we may really want to ask the valve: Are you all right? Are you lying to me? Have you been damaged (hung up, etc.)? These questions require either more sensory information, or a more complex system that can make inferences about the valve. If the system is more complex, we probably just want an answer, though we would wish to query it concerning the information that it is basing its inferences on. If we are in a hurry, and if we trust the inference system, the *higher level of abstraction* will probably be preferred.

In discussing matters with machines, higher levels of abstraction in the semantics of our discussion always imply arrangements of somewhat complex inference by the machine or its immediate monitoring units. For

---

[13] The sense is this: to the non-sentient system, the movement in not a completed sign, though it is a complete sign (with interpretation and actualization) to us and to others who may observe us turning the valve.. If the plumbing were sentient, for example by having sensors and a computational unit suitably programmed, the same behavior would be not just effective, but would also have informational value to that system also.

example, what we may want to ask the machine is really something like: Are all those valves shut? Or: are the valves open just the way they should be under the circumstances? If things are not as expected (whatever that is), then we may want to make finer-grained inquiries, say at the single valve-sensor level.

Similarly, the system may gain our attention only if certain configurations of the valves exist, or if indicators elsewhere suggest a problem. In either case, a fair amount of knowledge, and dynamic relating of machine states to inferences about intended goals, will be involved.

The cognitive *relata* of a Valve Language, then, could involve a knowledge base, and could accumulate experience. The patterns, and goals, can be complex and delicate. (People working with Space systems are keenly aware of the complexities of systems of valves, some Earth-based, some that fly.) The problems of sensor reliability, fine timing decisions, and so on, are real enough.[14]

Real systems of valves, of course, have different patterns of communication depending on their state of development. A human will ask different kinds of questions of the device when it is being checked out in the testing lab, than when it is functioning in its intended mission. For one thing, mission behavior may be very rapidly timed, or take place in remote locations, so that in moving from design and test phases to operational phases, there is a general shift from detailed human communication with small parts of a system, to internal-system communication among the various sensors, inference and control units, and related devices.

It is interesting that Valve Language, as discussed here, may not need to refer to the external environment. However, the local, device-internal environment must at least implicitly be handled in the computations that involve inference based on information from more than one internal position.

Information and Action. When the valves are turned on and off by

[14] Valve-heavy systems are important elsewhere, for example in the power industry.

remote means (certainly the case for the more complex systems of the above examples), *action statements* will be transmitted. Just as there are levels of abstraction in queries about the state of parts, subassemblies, or total devices, *there are levels of abstraction in imperatives for action.*

That is, each valve turner (or whatever part) could be addressed individually. But the human may not want to do this, or may not be able to because of the necessity for rapid or coordinated action of parts and units.

So, the human, or possibly a high level control device, will issue an action statement whose meaning has been set up to involve a certain pattern, perhaps even a pattern involving locally computed contingencies.

The higher level action statement is often just in the form of a specialized *releaser,* that is, it says "Do it now!", where 'it' has been specially set up.[15]

In other cases, high level directive to a machine may give more information about choices. For example, most sorts of directions that we imagine giving to ambulatory robots are of this kind.

Distal Languages and Contact Communication. A contrasting situation for communication with devices could involve what might be called a Nudge and Push Language.

As humans, we rely on spoken and written communication very heavily. Both visual and auditory senses are distal. Sometimes we do use closer means of communication with one another, and we may also do this with machines.

For example, suppose that a Space worker has one or more robotic assistants to help him or her do repairs in EVA. Such small devices may need to be shown what to hold[16], what to work on with a tool, how to

---

[15] We distinguish between releasers and specifiers in an action language. There are also qualifiers.

[16] Clamping, bracing, countermoving, and other stabilizing functions should be important for such semi-intelligent helpers. Consider how much such supportive functions are involved in ordinary shop work.

move the tool, and so on. The "showing", really a matter of nudging and illustrating by having the robot go into a semi-passive, semi-limp motor learning state, and being *moved,* then having initially arbitrary symbols or directives associated with the movement, might be done in the situation of actual repair, or in trial situations. A very adept helper would presumably build up a repertoire of motor-sensory understanding of such activities, given sufficient cognitive capability and memory.

The "showing" is not enough. The motor activity must be related to meaningful signals that can be communicated in the situations of repair. These signals may themselves be distal (e.g., spoken) rather than motor.

Other applications. Examples given above are meant to suggest potential embedding of language behavior, in the broad sense, in human-machine enterprises. Analysis of such situations must include, we would argue

- detailed analysis of what is to be done,
- explicit analysis of communications patterns (who and what will be involved in the communication),
- linguistic and semiotic analysis, both to suggest possibilities for communication, and to keep the means of communication manageable.

IV. Dynamic Communication Systems: Message-Passing in Object-Oriented Languages[17]

1. Introduction

We now proceed to an examination of object-oriented computation systems. Object-oriented systems are important to the question of action-related computation because their structure provides a partial model of external action systems. That is, certain computational entities, the objects, pass messages to the end of getting computation done. But there are differences between a strictly computational system, and a real machine/environment system.

In typical software systems, communication among system elements is well defined. In fact, defining the methods of communication between system components is a major part of traditional system design. In these cases, communication depends on static definitions. The problem is statically defined, as are the courses of interaction and the globally available data. However, if the system is essentially dynamic, with autonomous components of varying types and numbers entering and leaving the system, this type of system definition is inadequate. For example, consider a collection of robots permanently maintaining an outpost with limited human supervision. Such a system must tackle problems never envisioned by its designers, must dynamically organize itself to respond to novel problems, and must rely on explicit communication for shared information.

---

[17] The body of this section was written by Mr. Gordon Streeter.

The complete dependency of such systems on direct communication, rather than shared data or shared design, places heavy constraints on the unit of communication: the message. Each message must accurately identify its receiver, provide complete information as to its content, and cannot rely on any statically defined information about the sender or the receiver. In addition, the message must be a reliable means of communication. Provisions must be made for accurate transmission, comprehension, and execution of each message.

2. Scope

The following simple illustration is indicative of the kind of systems under consideration:

Robot A is one of a contingent of robots at work on a remote site which has little human supervision. A has the task of collecting debris from a construction site. A encounters a rock which is too heavy for it to lift alone. A needs help from some robot X. A sends the following messages in order to accomplish this:

X, come here.
X, lift end of rock.
X, carry rock to transport.
X, put rock in transport.

The problems associated with implementing a system of this type are numerous and cover a wide range of technologies. The implementation of a single robot of this nature is beyond the capability of current hardware and software technologies. The combination of these robots into a dynamic system introduces new problems of communication, priority-setting, and coordination. The current discussion concerns a fundamental aspect of

communication, that of message-passing. As such it will assume that the communicating components share a common language and that a physical layer supporting communications is present. In addition, the discussion will be limited to simple request-response communication, and will ignore the detail of any implied dialogues.

A three-element model serves for discussion of request-response exchanges. The three elements are: sender, receiver, and message. The roles of each element are evident from their names. A fourth element, dispatcher, will be added later. The following paragraphs use this three-element model for describing some of the problems involved in communication between dynamic systems of autonomous components.

There are several areas in which problems could arise in the above example. To begin with, a robot X must be identified in some way. Once identified, the robot may be unable to comply, either because it is immobile, or otherwise occupied. These problems could be determined before X actually enters the task by some pre-task discussion which irons out all that will be required. Other failures, however, could occur during the processing of the task. For example, X could experience an internal fault while carrying the rock to the transport, or could have a priority request from some other area of the site. None of these problems can be statically handled, since they can not be anticipated at design time.

3..Object-Oriented Languages

There is class of languages, called object-oriented languages (Saunders, 1989; Stroustrup, 1988), which support the request-response model described above. These languages are based on a design philosophy of completely encapsulated objects which request actions of each other by sending messages to each other. This design has four components:

Sender	-	the originator of the request
Receiver	-	the target of the request
Message	-	the packet containing the request
Dispatcher[18]	-	the mechanism for transmitting the message

Two languages, Smalltalk and Lisp Flavors, will be used for illustration. The two are fundamentally different in approach, but each implement full-featured object-oriented programming systems. The following paragraphs provide a brief introduction to object-oriented programming as well as to Smalltalk and Lisp Flavors.

Briefly stated, the process of software development in object-oriented languages is the creation of abstract data types (Ladd, 1989; Bailey, 1989). These types describe both the format of the data and the operations to be performed on the data. These are called "classes". Any given object is an 'instance' of its class. Class information is not available outside the class, and instance-specific information is not available outside the instance. The only way objects can interact is through message-passing. Message-passing is conceptually similar to, but fundamentally different from function invocation. With function calls, the caller accesses the target directly. Messages are sent via a message dispatcher, which identifies the actual code to be executed. This separation is important for simulating the kinds of autonomous systems discussed above.

Smalltalk (Goldberg and Robson, 1983; Saunders, 1984) is a complete object-oriented programming environment, originating from Xerox PARC, and

[18] Some definitions of object-oriented languages include languages which have encapsulation, but no message-passing. See, for example, Winston and Horn (1984) and Stroustrup (1988). These languages typically use function calls, rather than mesages. Ada, for example, would be included in such a definition.

closely tied with direct manipulation, mouse-menu interfaces. The system is based on a simple, single paradigm wherein every entity is an object, and objects interact via messages. The syntax of the language reflects this paradigm and has the general form of an object's name followed by a message to be sent to the object. The message may contain other objects as arguments. In the example below, an object of type 'List' is created, several strings are added to the list, the list is asked to sort itself, and the result of the sort is asked to print itself.

```
| aList |

aList := List new.
aList add: 'first string'.
aList add: 'second string'.
aList add: 'last string'.
aList sort print.
```

Lisp Flavors (Coral, 1987; Saunders, 1989) is an extension to Lisp, originating from a joint MIT/Symbolics effort. It is available in several of the Lisp dialects, including Symbolics Lisp and Allegro Common Lisp. Flavors inherits its syntax from Lisp, and in the Lisp tradition, gets its power from the complexity of its implementation, having many more options, modes, and parameters than Smalltalk. The terminology is somewhat different from Smalltalk as well. For example, Flavors objects are operated upon by "generic functions", rather than by "message-passing". While there are subtle differences in generic functions and messages-passing, to a large extent this is merely a difference in terminology, born out of the very different approaches of Smalltalk and Flavors. Flavors does employ a dispatcher to transmit messages, but since Lisp is a "functional programming language," this dispatcher is said to provide generic "functions". The following Flavors example (though not excellent Lisp)

performs the same function as the Smalltalk example above.

```
(prog (a-list)
    (setq a-list (make-instance 'list))
                (send a-list :add 'first-string')
                (send a-list :add 'second-string)
                (send a-list :add 'last-string)
                (send (send a-list :sort) :print))
```

The rock-carrying example above can be written in Smalltalk as:

```
X comeHere.
X liftEndOf: rock.
X carry: rock to: Transport.
X out: rock in: Transport.[19]
```

In Flavors:

```
(prog ()
                (send x :come-here)
                (send x :lift-end-of rock)
                (send x :carry-to transport rock)
                (send x :put-in transport rock))
```

4. Using Existing Implementations

While object-oriented languages provide a good platform for simulating the dependency of dynamic systems on explicit communication, they do not

[19] Note that capitalization is used in Smalltalk for two purposes. The first is to separate words in multi-word symbols. The second is to indicate the scope opf a symbol. Global symbols begin with a capital letter; local symbols do not. In this example, X and Transport are assumed to be global symbols, which rock is a local symbol representing the rock in question.

inherently provide for the kinds of problems found in dynamic systems with autonomous components. The following paragraphs describe possible methods of coping with these problems using Smalltalk and Flavors.

The first problem, that of identifying a possible receiver, is easily solved (or at least hidden) by placing the burden on the underlying communication software. The remaining discussion assumes that some lower level of software maintains a globally-available list of active communicants.

The problem remains, however, of discovering whether the intended receiver can respond to the message. One possibility is to keep a type entry in the list of active communicants. Each sender could then contain a database of the capabilities of receivers of each type. Before sending a message, the receiver would then have the responsibility to check the capabilities of each of the possible receivers until it found one which could respond to the request.

There are two problems with this approach. The first is that it places a heavy burden on the sender, requiring it to be aware of every capability of every possible type of receiver. If the number of different types of receivers were large, storing and accessing the database could unnecessarily complicate some simple components. This would be an even greater problem if the sender were a human. In a large system, the human could not be expected to internalize the database, but would have to rely on external, probably automated, storage.

The second problem is that the solution is most naturally a static one, in which the list of capabilities is created at implementation time. the solution does not lend itself well to systems in which the types of

participants are changing, or in which the capabilities of a single type may change over time.

The more natural solution is to have each component provide its capabilities on request. This would enable the sender to test candidate receivers by inquiring of each if it could comply to a given request if such a request were issued. Both Smalltalk and Flavors provide such capabilities. Smalltalk objects can be interrogated in this manner via the message "respondsTo:". For example, the message:

```
X respondsTo: #comeHere
```

would return true if X could comply, false if X could not. Flavors provides the message: ":operation-handled-p", which performs the same function.[20] Flavors supports a similar message, ":send-if-handles". This message is sent with another message as its argument. The receiver sends the argument message to itself if it determines that it can handle it. However, ":send-if-handles" provides no feedback as to whether the argument message was actually sent.

Using this method, the sender could select an appropriate receiver from a list of candidates and send a message to it. This could be implemented in Flavors something like:

```
(send (send candidates :responding-to ':come-here)
                      :come-here)
```

Even in this compressed form, with the "responding-to" method doing the

[20] Note that both "respondsTo:" and "operation-handled-p" indicate whether the object responds to a message of the given name. No semantics is implied. For example, integer objects and list objects both respond to the message "add:", but the message has different meanings to each type.

work of searching out an appropriate receiver, the process is clumsy. Also, it does not allow for the condition when no receiver is able to respond.

Another extreme would be to force the receiver to respond to all requests. Through a process which might be called "smart-reception", the receiver must accept all messages and acquire the expertise to handle them appropriately. There is a single point in both Smalltalk and Flavors at which such an action could take place. In Smalltalk, when an object is sent a message for which the dispatcher can find no response, the dispatcher sends the object the message: "doesNotUnderstand:" with the original message as its argument. The Flavors dispatcher reacts similarly, using a message called: ":unclaimed-message". In addition, Flavors provides the ability to supply a ":default-handler" which, if present, is used as the handler for messages for which no explicit handler can be found.

Having this capability, however, is a long way from solving the problem of smart reception. The receiver may simply not have the information or the hardware required to solve the problem. For example, if a one-armed robot is sent the message: "hang-wall-paper", the message will most likely end up in the default message handler. Even if the robot could correctly interpret the message, it could not respond without changing its "physiology".

Though not practical on a large scale, some degree of smart reception is essential to handle small discrepancies. For example, it is likely that a robot equipped collect samples from the ground could physically respond to a request to dig a small hole, even though it was not specifically designed to do so. The sender could not be expected to detect this ability; only the receiver has the knowledge required to do so.

Neither of these approaches deal with the condition in which the receiver

attempts to respond and then fails to do so. This situation could arise when the receiver experiences a malfunction, or when the task itself changes in some unexpected way which invalidates the current task description and forces the receiver to request more information. In both conditions, the receiver must notify the sender of the difficulty. Unfortunately, neither Smalltalk not Flavors provides the receiver with the identity of the sender. Both languages provide the variable "self", which is the identity of the currently acting object, but neither support the concept of "sender".

This problem could be solved by a programming convention which forces the identity of the sender to be sent with every message. In Smalltalk, this convention, combined with the receiver selection process described above might look like:

```
(Candidates respondingTo: #comeHere)comeHereWithSender: self
```

5. Extending the Dispatcher

Message-passing which follows these conventions is much more complex than the simple messages described above. This overhead complicates the coding process to an unacceptable degree. Fortunately, these two functions can be delegated to the dispatcher, relieving the sender of such a burden.

Moving this work to the dispatcher makes sense for two reasons. The first is reducing code bulk and complexity. If every message requires that the actual receiver be selected from a list of candidates, then the function can be inserted once, in the dispatcher, rather than many times in each sender. The second reason is that the dispatcher knows the identity of both the sender and the receiver, and thus can provide the receiver with the identity of the sender.

To accomplish this means to change the dispatcher. In Smalltalk, this is not an easy task, because the dispatcher is part of the "byte-coded" interpreter, and source is not available for it. Flavors does not have such a restriction. However, the Flavors code for the dispatcher is quite complex, so the following simple example, taken from Winston and Horn, 1984 (pg. 245), will be used for illustration.[21]

```
(defun send-message (target method &rest arguments)
  (apply (get (get target 'is-a) method)
         (cons target arguments)))
```

The example corresponds closely to the Flavors message-passing shown in previous examples using "send". The terminology is different, however: "target" is used for "receiver", and "method" is used for the message name. The example assumes that the property list of the receiver's type (accessed through "is-a") contains the code to implement the message, filed under the name of the message.

The first step in expanding on this example is to provide the receiver with the identity of the sender. The dispatcher can accomplish this by providing "self" to the receiver under the name "sender", as follows:

```
(defun new-send (receiver message &rest arguments)
                (apply (get (get receiver 'is-a) method)
                       (cons self (cons receiver arguments))))
```

This allows the "come-here" code of some robot X to be written as:

```
(defun come-here (sender self)
```

[21] Winston and HOrn (1984) also present a more complex example, supporting the concept of "befores" and "afters", which is somewhat close to an actual Flavors implementation.

```
(new-send self :go (new-send sender :where-are-you)))
```

The second step is to provide the new sender with the ability to search a list of candidates and send the message to the first acceptable candidate. This task is performed by a helping function:

```
(defun send-help (rec-list message)
 (cond ((null rec-list)
        (get (get self 'is-a) ':failed))
                    ((get (get (car rec-list) 'is-a) message))
       (t (send-help (cdr rec-list) message))))

(defun new-send (receiver message &rest arguments)
                  (apply (cond ((atom receiver)
                 (get (get receiver 'is-a) message))
                (t (send-help receiver)))
           (cons self (cons receiver arguments))))
```

With this dispatcher, the receiver may be specified as a single object, or as a list of objects. If a single object is provided, the message is sent to it directly. If a list is provided, the list is searched for an object which can respond. The first such object is used as the receiver. If no receiver is found, the sender is sent the message ":failed". This technique allows A to send the message ":come-here" to an unknown receiver, as shown below.

```
(new-send candidates :come-here)
```

6. Conclusion

Dynamic systems pose many special communication problems. While object-oriented systems provide a means of emulating the way such systems must communicate, they do not intrinsically solve those special problems. The technique described above solves the problems of identifying receivers and making the identity of the sender available to the receiver. But this technique is useful only for the simplest of problems, and must be expanded to be useful in tasks which require ongoing dialogues. In addition, techniques need to be developed to support "smart reception" of messages. However, none of these techniques will prevent the complete failure of all possible receivers to respond to a message. This responsibility must rest with the sender of the message.

V. Conclusions and Issues

Conclusions

This report has discussed dimensions of action languages for communication between humans and machines, and examined in some detail the message-handling capabilities of object-oriented programming systems.

Design of action languages is seen to be very contextual. Economical and effective design will depend on features of situations, the tasks intended to be accomplished, and the nature of the devices themselves.

Current *object-oriented systems* turn out to have fairly simple and straightforward message-handling facilities, which in themselves do little to buffer action or even in some cases to handle competing messages. Even so, it is possible to program a certain amount of discretion about how they react to messages. Such 'thoughtfulness' and perhaps relative autonomy of program modules seems prerequisite to future systems to handle complex interactions in changing situations.

Issues

Description and understanding of what is in situations, and what may suddenly happen within them, emerges as critical for the development of language-mediated communication about action.

Description is problematic, understanding by machines is difficult[22], and work on incorporating situational features and events within language systems is not new but is still hardly a major pursuit of linguists.[23]

[22] Sometimes, too, by humans.

Accounting for the unexpected in real situations is part of the capability of any organism, but is not well handled in computational systems, perhaps because of their necessity for specifying everything exactly for computing machines as we are familiar with them.

In robotics, there has been a schism for several years between AI researchers who wish to plan everything in some detail, and those who feel that planning requires needlessly complex and perhaps inadequate response to actual situations. This controversy is not especially relevant to the manufacturing applications involving repeated operations on fixed parts, but it becomes important rapidly when dealing with ambulatory robots.

Further analysis of action language would involve, as has been discussed above, more details of just how information changes, and how surprises come about, to be accounted for or ignored, during the course of action.

Action systems should be compliant if we build them, but have to be autonomous, at the very least in order not to occupy our attention unduly. Autonomy may be hazardous, though. It is not clear that technology planners have adequately faced this issue.

Various linguistic issues bear on the design of restricted action languages.

One design issue is just how complete the language design should be. That is, we know that in the case of human languages, some parts of them change to meet new situations, or even because human cultural systems seem to maintain a balance of persistence and modification.

If humans are involved in the machine communications, they may feel a need to modify the language in some ways. Perhaps only vocabulary could be added or changed. This could be a problem since other users might not be up with the latest vocabulary.

---

[23] The most developed treatment of situational factors in linguistics and logic is in Barwise-Perry situational logic. But it is only one possible approach.

Suppose, though, that the language could not change at all. Then special references to unique situations might not be easy to express. Further, exchanges involving any kind of abstraction or convenient generality (or even vagueness) might also be difficult.

The planned versus evolutionary dimension of language planning is reminiscent of the controversy among AI researchers in robotics. Perhaps it is true that a static language will always be inadequate, but setting bounds on variation may be needed in some cases, especially with machines which have meager cognitive capabilities.

What kind of language to use provides many linguistic questions. Since humans will be communicating some times, they could feel more comfortable if features of human languages were incorporated into the human-device communication system. But what features, and from what languages?

A given language, say English, has many structural features.[24] Surveys of other languages show some of the same, and some different ways of putting verbal signs together.[25] Particularly interesting are those languages that have been influenced by a number of sources, or that serve as trade languages.[26]

Since human language processing by machine is difficult, it may be that more graphic "languages" such as schematized pictures, perhaps augmented by sounds, are most promising to pursue, at least for some tasks.

---

[24] See, for example, the discussion in Huddleston's *Introduction to the Grammar of English* (1984), which treats familiar phenomena of English in theoretical and structural terms, at a very fine level of analysis.

[25] One discussion involving examples from many languages is Haiman (1985). Grammars and tutorials of various languages provide ready information, also.

[26] Such as Swahili. Languages of islands where trade has been important are sometimes relatively simple syntactically and are thought to be easy to learn. Hawaiian, Indonesian come to mind. English is now, of course, very widely spoken. But, any human language seems to have its own peculiarities, its own complexities, and of course, its own adherents.

Communicating with an ambulatory robot would probably be well handled with a picture language, at least in part.

Another possibility, useful perhaps for languages where query is more important than depiction, is to take a well-proven notion in the analysis of human language such as that of case relations[27], then to restrict syntactic possibilities.

So the action language concept raises many possibilities for realization and many research issues. The history of computation over the past decade or so has shown that models for computation often follow closely available machines. Actor and agent computing models, which are more restricted than their names would suggest,[28] are just about what one would need to reflect computation using multiple processors working on the same problem, with serial communication in networks. As devices become available that are autonomous but connected, it seems likely that computational means will be modified. But history also shows that this trend is sometimes sluggish, so that there is often a tendency to use older models with machines that have more challenging and complex tasks. The analysis offered in this research report is aimed at counteracting that trend, and facing issues of optimal language and computation design based on an examination of the situations of use.

Daniel G. Hays, Gordon Streeter
The University of Alabama in Huntsville
Spring/Summer 1989

---

[27] Fillmore called the attention of linguists to the case notion very persuasively some time ago (1968). In computation, Roger Schank has made good use of case assignments.of natural language terms from his early career (Schank and Abelson, 1977).. An interesting recent presentation of case-like information in lexical organization is by Judith Markowitz (1988). John Sowa's work in propositional calculus models, or conceptual graphs, continues and extends the notion (1984).

[28] See the comments in Tello (1989) for a succinct description of the capabilities. Agha's book (1986) also contains thorough discussion of this kind of model.

References

Agha, Gul, *Actors: a Model of Concurrent Computation in Distributed Systems,* MIT Press, 1986.

Bailey, Stephen C., "Designing with Objects," *Computer Language,* Volume 6, No. 11, January 1989, pp. 34-43.

Coral Software, "Release Notes for Allegro Common Lisp Flavors 1.0", November 25, 1987, Coral Software, Inc.

Fillmore, Charles, "The Case for Case", in E. Bach and R. T. Harms, Eds., *Universals in Linguistic Theory,* Holt, Rinehart and Winston, 1-88, 1968.

Goldberg, Adele and Robson, David, *Smalltalk 80: The Language and its Implementation,* Addison-Wesley, 1983.

Haiman, John, *Natural Syntax: Iconicity and Erosion,* Cambridge University Press, 1985.

Hewitt, Carl, "Viewing Control Structures as Patterns of Passing Messages", *Journal of Artificial Intelligence,* Vol. 8, pp. 323-364.

Hewitt, C. and DeJong, P., "Analyzing the Roles of Descriptions and Actions in Open Systems", *Proceedings of AAAI,* 1983.

Huddleston, Rodney, *Introduction to the Grammar of English,* Cambridge University Press, 1984.

Ladd, Scott Robert, "Designing with Class," *Computer Language,* Vol. 6, No. 11, April, 1989. pp. 81-87.

Markowitz, Judith, "Graded Set Membership", in M. W. Evens, Ed.,

Relational Models of the Lexicon, Cambridge, 1988, pp. 239-260.

Rembold, Ulrich and Hörmann, Klaus, *Languages for Sensor-Based Control in Robotics,* Springer-Verlag, NATO ASI Series,1987.

Saunders, John H., "A Survey of Object-Oriented Programming Languages," *Journal of Object-Oriented Programming,* Vol. 1, No. 6, March/April, 1989.

Schank, Roger and Abelson, Robert, *Scripts, Plans, Goals, and Understanding,* Lawrence Erlbaum, 1977.

Sowa, John F, *Conceptual Structures: Information Processing in Mind and Machine,* Addison-Wesley, 1984.

Stroustrup, Bjarne. "What Is Object-Oriented Programming?", *IEEE Software,* Vol. 5, No. 3, May, 1988.

Tello, Ernest R., *Object-Oriented Programming for Artificial Intelligence,* Addison-Wesley, 1989.

Winston, Patrick Henry and Berthold, Klaus Paul Horn, *LISP,* Second Edition, Addison-Wesley, 1984.